MATLAB in HPC

SPMD and Distributed Arrays.

Anders Sjöström LUNARC
anders.sjostrom@lunarc.lu.se

SPMD

Overview

- `spmd` (Single Program Multiple Data)
- `labindex` and `numlabs`
- Exchanging data between workers explicitly
- Data transfer to the client using composite arrays

parpool

- Similar to `parfor`, `spmd` requires a parpool in order for code to run on workers

- If a `parpool` doesn't exist, one will start if that is the default behavior

`spmd` (Single Program Multiple Data)

- Code inside `spmd` blocks run on all workers
- Unlike `parfor`, variables maintain state between calls to `spmd` as well as in `parfor`
- Can be used for loading data to be used in `parfor` loops

```
spmd
    % myfile.mat needs to be available on workers
    data = load('myfile.mat');
end

parfor I = 1:N
    % loop using data
end
```

`labindex` and `numlabs`

- Helps control what is executed on a worker
- Inside a spmd block
  - `labindex` returns the rank of the worker
  - `numlabs` returns the total number of workers in the pool

```
spmd
    switch labindex
        case 1
            % Code for worker 1
        case 2
            % Code for worker 2
        ...
    end
end
```

Create a different array on each of the workers

```
>> magic_squares
```

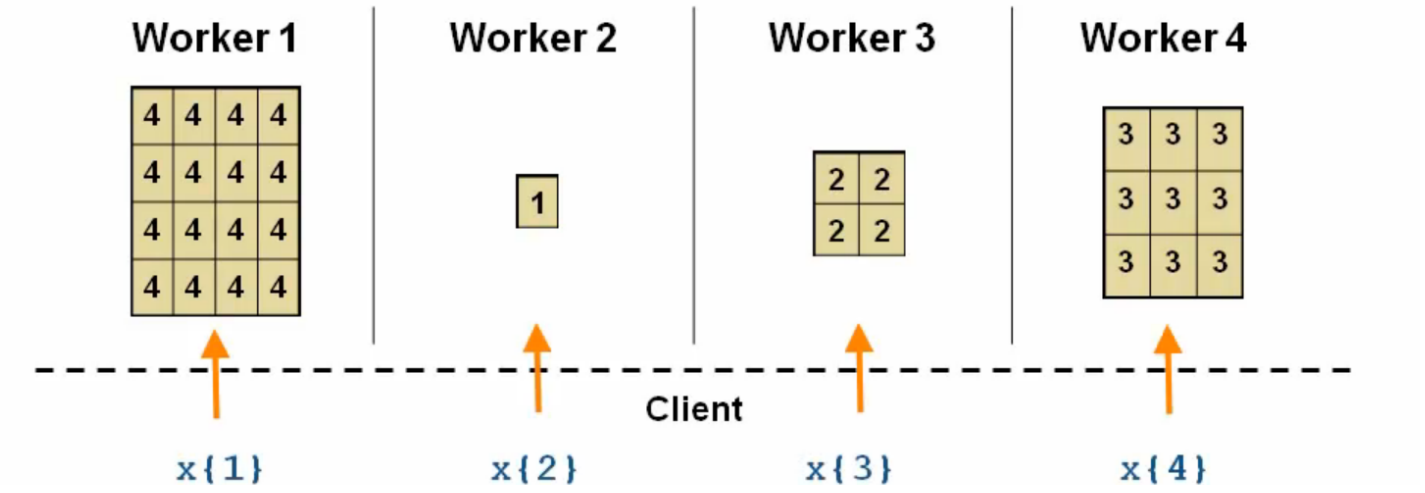
```
spmd
    % Build magic squares in parallel
    m = magic(labindex + 2);
end

for ii=1:length(m)
    % Plot each magic square
    M = m{ii};
    figure, imagesc(M);
end
```

```
>> approx_pi
```

```
quadpi = @(x) 4./(1 + x.^2);

spmd
    a = (labindex - 1)/numlabs;
    b = labindex/numlabs;
    fprintf('Subinterval: [%-4g, %-4g]\n', a, b);
end

spmd
    myIntegral = integral(quadpi, a, b);
    fprintf('Subinterval: [%-4g, %-4g]   Integral: %4g\n', ...
        a, b, myIntegral);
end

spmd
    piApprox = gplus(myIntegral);
end

approx1 = piApprox{1};   % 1st element holds value on worker 1.
fprintf('pi            : %.18f\n', pi);
fprintf('Approximation: %.18f\n', approx1);
fprintf('Error         : %g\n', abs(pi - approx1))
```

Composite Arrays

- Composite: client-side data-type for viewing data on the workers
- Outside of `spmd`, index with () or {} to get the data of one of the workers to the client



```
>> spmd, x = labindex/numlabs, end
Lab 1:
  x =
      0.5000
Lab 2:
  x =
       1
>> x
x =
   Lab 1: class = double, size = [1  1]
   Lab 2: class = double, size = [1  1]
>> y = x{1}
y =
    0.5000
>> whos x y
  Name      Size             Bytes  Class        Attributes

  x         1x2                697  Composite
  y         1x1                  8  double
```

Types of Composite Arrays (non-distributed arrays)

- Replicated

```
spmd, x = numlabs, end
```

- Variant

```
spmd, x = labindex, end
```

- Private

```
spmd, if labindex==1, x = rand, end, end
```

Limitations

- The body of an `spmd` statement must be transparent

```
spmd
    load x
    y = x;
end
```

```
spmd
    data = load(['X' num2str(labindex)]);
    y = data.x;
end
```

Distributed Arrays

Overview

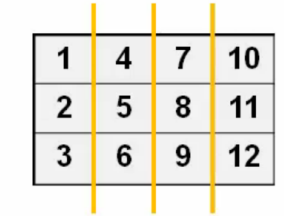
- Distributed Arrays
- Constructing Distributed Arrays
- `distributed` and `codistributed`
- Working with Distributed Arrays

parpool

- Similar to `spmd`, `distributed arrays` require a parpool in order for code to run on workers

- If a `parpool` doesn't exist, one will start if that is the default behavior

Distributed Arrays



- One variable, split over multiple workers
- However, the MATLAB client sees the variable as one
- Mainly of interest with a cluster, combining the memory of multiple machines
- If the function has been overloaded for distributed arrays, there should be minimal changes to the code

Creating Distributed Arrays (1)

- Matrix creation functions have been overloaded for distributed arrays
  - `zeros(...,'distributed');`
  - `randn(...,'distri`

```
Z = zeros(4,4, 'distributed')
```

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

```
A = reshape(1:16, 4, 4)
A = distributed(A)
```

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

- If a variable has the same value on all of the workers, use `distributed` directly

Creating Distributed Arrays (2)

- Use case: creating a large matrix from multiple files or one large file would not fit into the memory of one computer
- Create data on each worker
- Combined into a distributed array using `codistributed.build` and `codistributed1d`

- Specify the size of the distributed array and optionally the partitioning

Working with Distributed Arrays

- A collection of MATLAB functions are overloaded for `distributed` arrays

- Overloaded functions can be called similar to other data types (e.g. numeric)

```
>> methods distributed
```

```
abs
acos
acosd
acosh
acot
acotd
acoth
acsc
acscd
acsch
all
and
angle
any
...
```

- Call `gather` to convert back to a numeric array

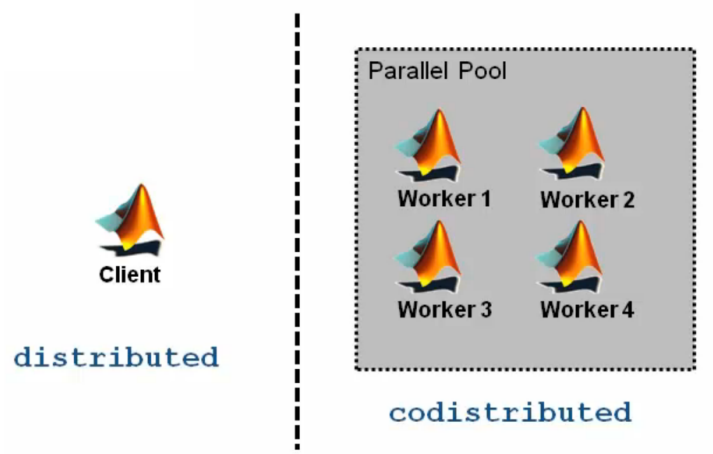
Using Distributed Arrays on Workers

```
>> distrib_example
```

```
W = ones(8,'distributed');  % Distribute to the workers
spmd
    T = W*2; % Calculation performed on workers, in parallel.
             % T and W are both codistributed arrays here.
end
T            % View results in client.
whos         % T and W are both distributed arrays here.
```

`distributed` and `codistributed`

- The same distributed array will have a data type of:
  - `distributed`: on the client
  - `codistributed`: on the workers (within a `spmd` block)



Using Codistributed Arrays on Workers

```
>> codistrib_example
```

```
p = parpool(2);
spmd
    % Define 1-D distribution along the 3rd dimension
    % 4 parts on worker 1, and 12 parts on worker 2
    codist = codistributor1d(3,[4,12]);
    Z = zeros(3,3,16,codist);
    Z = Z + labindex;
end
Z % View results in client (distributed array here)
```

LUND
UNIVERSITY